

Experiences in Software Testing Education: Some Observations from an International Cooperation

Joseph Timoney, Stephen Brown

Dept. of Computer Science, NUI Maynooth, Maynooth, Co. Kildare, Ireland

jtimoney@cs.nuim.ie, sbrown@cs.nuim.ie

Deshi Ye

College of Computer Science, Zhejiang University, Hangzhou, China

yedeshi@zju.edu.cn

Abstract

This paper examines the outcomes of teaching a course in Software Testing in Ireland and China over a two-year period. In both institutions the delivery of the course is constrained to two-week duration. The learning objectives for this course are explained. The outcomes of the courses in terms of student learning are compared and analyzed. A number of observations are made that lead to recommendations as to how the educational experience can be improved for the students in both countries.

Keywords: Software testing education, international education collaboration, software engineering curriculum

1. Introduction

Software Testing is a key component of any degree in Software Engineering and is an essential part of the skillset of graduate Software Engineers. It has been estimated by the US department of Labor that the demand for software engineers, will grow by 38% between 2006 and 2016, which is much faster than average for all other occupations. [1]. Moreover, the demand by industry for software testing skills should be increasing as recent surveys indicate that the economic impact of inadequate testing is still a major issue for them [2], [3]. From the research community, Software testing education has received more attention over the last 10 years with analyses ranging from the broad strategic overview, [4], to the specifics of course structures and lecture delivery, [5], [6] and [7]. However, this also implies that the teaching of software testing is not as mature as other fields, for

example unlike many subjects in computer science. Thus, more observations followed by analyses need to be carried out on the teaching process to understand how best to present this topic to students inexperienced at software development, to organize the balance of activity between theoretical and practical, and how to adapt or focus the breadth of delivery in the face of resource-based constraints.

This paper attempts to contribute by reflecting on the experiences gained in teaching software testing to students at the dept. of computer science, NUI Maynooth in Ireland, and at the College of Computer Science, Zhejiang University (ZJU), Hangzhou, China. The next section will describe the course structure, the student cohort, all the features of the teaching schedule, and the lecture environment in both countries. After this, results of an analysis of the examination scores by the students will be presented. This will endeavor to disassociate regional issues in the teaching process and also highlight more general observations. This will then lead to a set of recommendations that have implications for the broader community that hopefully will stimulate future developments by the pedagogical community at large.

2. Course Structure in both Colleges

Software Testing is taught at the Dept. of Computer Science in NUI Maynooth (NUIM), Ireland as part of a Masters Degree in Software Engineering (MsSe), while at ZJU it is a subject in their undergraduate degree in Computer Science (CS). In the last few years a number of initiatives have been made to forge teaching and research links between the two Universities. The delivery of the module in Software Engineering for the College of Computer Science at ZJU by a lecturer from

the NUI Maynooth is one such collaboration. The responsibility of the lecturer is to prepare and present course material, and also assess the students by setting and marking an exam paper at the end of the academic term. The course is delivered in English. Table 1 presents the details for both institutions of the student cohort for this module, the schedule of teaching, and the balance between project work and the written exam in its assessment.

Table 1. Features of the software testing module delivery at both universities

	NUIM	ZJU
Student Seniority	1 st year postgrad	3 rd yr undergrad
No. of students	10-20	160-180
Nationality	International	Chinese
Teaching Period	2 Weeks	2 Weeks
Lecture Hours	12	21
Lab Hours	12	3
Timetabled Assignment	24	NA
Course Assessment	Written Exam 60% Practical 40%	Written Exam 60% Practical 40%

From Table 1, the students taking Software Testing as part of the MsSe qualification at NUI Maynooth are first year postgraduates. To qualify for entry for this course they must have a degree in Software Engineering or a related discipline, and must have scored 60% or greater in their degree. This ensures that the programming abilities of the students should be strong. ZJU's high ranking within China ensures that entry to its undergraduate programs is very competitive and means that the students are of high caliber. CS Students at ZJU study Software Testing in the third year of a four year degree program. Prior to this they have taken courses in Procedural and Object-orientated programming. The number of students on the MsSe course at NUIM is compact, with only between 10 and 20 on the course, while on the CS course at ZJU the class is given to approximately 170 students. This has an effect in terms of lecturer management of the student body. The students of the MsSe course come from a variety of both educational and international cultural backgrounds. In general the proportion of native to non-native students is 50/50. English is most often not the native language for the non-Irish students. On the CS course at ZJU, all the students are of

Chinese nationality and are drawn from many of the different regions around China. Having had instruction in the English language since high school that continues after coming to college, their comprehension and communication abilities are good.

In both cases the course duration is two weeks long. For the MsSe course, the first week is mixed between lectures and practical laboratories. These are given in an intensive manner, meaning that for each day the student will have approximately 5 hours of classes. In the second week the students individually work on a project that must be submitted by the end of the week. This is supervised but not continuously. At ZJU, the students have 21 hours of lectures that are divided evenly over the two-week period. They also have a lab session each week that is one and a half hours long. The lab sessions are supervised. At NUIM, the students are taught in a dedicated lab for the students so that during lectures they have access to a computer, while at ZJU, because of the class size, lectures are given in a large classroom. Finally in both cases, the assessment mark is a weighted combination of a mark for the students' practical work and the score on an end-of-term written examination. At NUIM, the weighting is 60/40 between the written and the practical which is the same as at ZJU. Both written examination papers have problem-based questions and theory-based questions. In setting both papers the aim is to give an even balance between the two question styles. In the exam paper for the ZJU students the problem-based and theory-based questions are distinct while at NUIM this is not the case.

The teaching material in both universities is similar, however, at Master's level the subjects are treated with more depth. Within the two-week period, the learning objectives for both courses are:

- 1) Introduce the history to the field of Software Testing, its financial relevance to industry and describe some of the well-known software failures due to inadequate testing. This is to motivate students for the need for software testing.
- 2) Define and explain many of the terms associated with Software Testing. This includes the key concepts of Black-box and White-box testing, and the specific application of these methods in Unit testing, Integration testing and System testing.
- 3) Using a suitable example specification and program, explain in detail how various testing techniques for white-box and black-box testing are applied. Discuss the relative merits and drawbacks for each method. The inclination here is towards Procedural programming but Object-Oriented

programming is also addressed. Tools for implementing tests are also introduced.

4) Explain how testing can be incorporated within the software development lifecycle, and the emphasis given to testing in particular methodologies. Discuss how testing is carried out in industry.

5) Mention the successes and difficulties in automating software testing. Additionally, provide an overview of the directions of research in software testing.

There is no one course textbook on which the material is solely based, instead it is drawn from a variety of different sources.

An analysis of the examination results from both courses can allow some insight into how these learning objectives are achieved. Figure 1 shows a scatter plot of the percentages scored by the students of ZJU for the examination against those of the practical assignment for 2007. The data is plotted using asterisks, while the dashed diagonal indicates the point of linear relationship between the two axes.

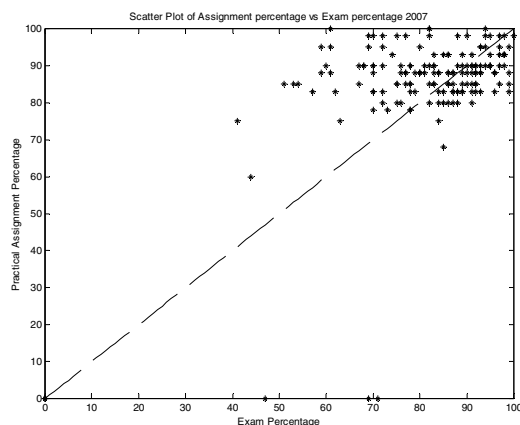


Figure 1. Scatter Plot of the Exam score percentage versus the Practical assignment score percentage by ZJU students in 2007.

The data is mostly in the upper right hand corner of the plot. There is approximately an even number of data points either side of the diagonal. However, the data above the diagonal is more diffuse compared to that underneath. This indicates that about half the class performed just marginally worse in their practical assignment than on their exam paper. Most of the remaining students did between in their practical assignment but the spread of marks was over a larger

range, between approximately 60-100%. Thus, in general a good examination mark indicated a closely valued practical mark but a poorer examination mark did not always mean that the practical assignment mark was as low.

Figure 2 shows a similar scatter plot for the NUIM students collated from the years 2007/2008. The number of students was much smaller. Again similarly to ZJU, there is approximately the same number of NUIM students above and below the diagonal. However, the data points appear to be spread over the range 60-90% on either side of the diagonal, but with most being reasonably close to the diagonal. This suggests that in general the marks for the examination will mirror those of the practical assignment but that there are sufficient outliers to say that this is not always the case. Students appear to be more likely to do better in the exam than the practical assignment.

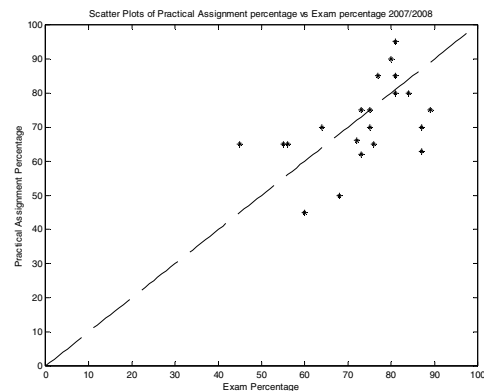


Figure 2. Scatter Plot of the Exam score percentage versus the Practical assignment score percentage by NUIM students in 2007/2008.

Overall, there is a reasonable similarity between the students' results in both countries to say that they are learning both the theory and practice of software testing in equal measures. Therefore, the course structure is reaching its stated learning objectives in this international context.

3. Observations on the learning objectives

In setting out the learning objectives it was realized that teaching this course in the constrained time period has, by necessity, required a number of compromises in terms of what could be taught from the whole field of

Table 2. Teaching difficulty of topics within the learning objectives.

Learning Objective	(1)	(2)	(3)	(4)	(5)
Conceptually straightforward	Yes	Yes	No	Yes	No
Single textbook available	No	Yes	No	No	No
Programming experience required	Desirable	Desirable	Essential	Essential	Desirable
Software Engineering experience required	Desirable	Desirable	Desirable	Essential	Highly Desirable
Ease of demonstration and generalization	Yes	Moderately difficulty	Difficult	Difficult	Difficult

software testing. Without question, a software testing course must have a practical dimension, however, the introductory motivation for testing cannot be ignored either, otherwise young students in particular that are inexperienced at programming may not see the relevance of this subject and not take it sufficiently seriously. The constraints mean that the whole field of software testing must be divided into the essential and the desirable areas to cover. Furthermore, within the essential areas some topics have been found to be easier to teach because of the widespread agreement across the literature as to what they mean and how they should be approached. Table 2 illustrates our teaching impressions and criteria for the learning objectives given in Section II.

For Learning Objective (1), the history and background of software testing are very interesting and not overly complicated. The sheer scale of some of the catastrophes given such simple causes is fascinating for many students. Information about these must be gathered from a variety of sources. One problem is that studies of the financial impacts of software testing age quickly. Furthermore, the results of these studies may be concentrated in a particular sector of the software industry so that it is difficult to present how software testing has impacted the variety of software business organizations comprehensively. A deeper knowledge of the cost-benefit relationship of software testing may become very important if students pursue careers in the area, and reach the level of project manager [8]. In this capacity they may be called upon to create budgets for testing. Knowledge of Software Engineering (SE) practices at this stage may help to underline its relevance. Actual demonstrations can be made of a

software program with errors and a broad generalization of the principles can be given.

Providing the core definitions of Software testing can be done in two ways: either using one accepted textbook or by fusing together the material from a number of texts such as [9], [10], [11], [12] and [13]. One issue for students is that the various texts are sometimes individualistic in their definitions, and work is required to resolve this into a more unified coherent form. Unfortunately, no definitive ‘Theory of Software Engineering’ exists unlike the theoretical concepts that underpin other fields in computer science that are more amenable to mathematical descriptors [8], [14]. Again, prior knowledge of programming and SE practices assists in quickening the learning process. Simple demonstrations of Black box and White box testing can be done but any attempt to show how they are applied beyond Unit testing takes more effort, skill and care. Regarding the variety of testing methods, they are not always easily explained. Although many of the textbooks are excellent in their coverage and clarity no one textbook gives a complete treatment of suitable testing methods for both procedural and object-orientated programs. The most comprehensive text in terms of the number of testing methods for procedural programming is [15]. However, this book may not reflect current realities so well. The example employed by [15] is written in Pascal and no treatment of object-orientated programming is given. No exercises for the student are given either. Other books that consider such examples as the well-known ‘Triangle program’ do not contain such breadth [13]. Additionally, the actual algorithm for creating the Triangle is more difficult for students to comprehend than [15] example program. For the testing of object-orientated software, other

textbooks are available such as [16], but are specialized and are not ideal for teaching a short course because of the complexity and level of detail [17]. Ideally, an integrated textbook would be best for teaching our type of course.

Testing methods are impossible to understand by students without programming experience, and their depth of programming practice is also a factor. Students who have had little opportunity to develop and debug their own programs have less appreciation for testing methods. Furthermore, this also applies to those who have had little exposure to programming complex conditional structures and loops. Knowledge of software engineering does help as a motivator. Demonstration of the testing methods can be difficult, particular when trying to advance the principles of application. Within short duration courses students may not always have sufficient time to fully understand how to implement these methods for new, previously unseen problems. Other issues that arise when teaching testing methods is rigor when creating and interpreting specifications, how to decide on which combinations of methods to use for a particular type of program [8] and how to create testable software, which is also a topic for a programming course [4], [17].

The application of software testing within the SDLC is also difficult to cover. No good textbook is available for this topic. The theory is much easier to teach than the actual practice. This is hampered both by the idiosyncratic approach to the SDLC within industry and in different international contexts [18]. Furthermore, it is very difficult to create programming assignments that can really mirror in terms of complexity and duration how testing fits into software development [19]. Tools have been suggested, [7], but their incorporation within a detailed textbook would be highly desirable. Student experience in programming and software engineering is also important for the true understanding of the relevance of the theory.

The final topics of the automation of software testing and research directions are more difficult to teach at undergraduate level than postgraduate level, particular the research component. Again no textbook is available. Both of these are not easy subjects to include in a textbook as even within a short space of time any automated tools or promising research directions can lose their currency. Teaching how to implement automation practically requires knowledge from fields other than software testing, such as software modeling, which students may not have facility with. The same is true for teaching the state-of-the-art in the research field as many ideas are usually brought together in the literature to create new approaches.

To overcome many of the hurdles encountered, some recommendations can be offered that would be of

much benefit to the students and may also help many educators when designing new courses in the field, and industry practitioners.

- Develop a flexible Cost-benefit analysis strategy for software testing [8]
- Create a consolidated and standardized set of the fundamental principles of software testing [8], [14]
- New textbooks should expound specification languages for software testing, inclusive of Model-based testing and the more frequently used written specifications [8].
- A textbook that amalgamates procedural and object-orientated testing in a clear, concise manner is needed. Examples should demonstrate the effectiveness and shortfalls of each test technique. It could also address the merits of various combinations of test techniques
- Introductory programming modules could help to incubate an appreciation for software testing by including instruction in writing testable software [4]
- All aspects of the integration of software testing into the SDLC, whatever the process being employed, would benefit from more support examples and tools.
- A textbook treatment of the practice of the automation of software testing would be helpful
- More survey results on the diversity of international industry practice in software testing would be very informative [18], [20], and [21].

4. Conclusion

This paper has examined the teaching of a compactly structured course in software testing in an international capacity. Analysis of the relationship between the scores in the examination papers and the practical assignments for the students was carried out. The results showed that the balance of understanding was similar in the two geographically and culturally separate countries giving a high level of confidence that this course is suitably structured for the students. Although this course structure allows us to achieve our

learning objectives our observations from the teaching process led to a number of recommendations, many of which correlate with the literature. Fulfilling these would enhance its delivery and also enrich the student learning experience.

Future work will implement some of the recommendations given in the near future and will carry out a more detailed analysis of student marks to assess their impact. Student feedback will also be incorporated into the measurements to boost the significance of this analysis

References

- [1] US dept. of Labor, Bureau of Statistics, *Computer Software Engineers*, <http://www.bls.gov/oco/ocos267.htm>
- [2] J. Everett, "Software Testing hits the big time", *IBM, presentation*, <https://www304.ibm.com/jct09002c/university/students/jobs/geverett-swtesting.pdf>
- [3] NIST, "The Economic Impacts of Inadequate Infrastructure for Software Testing", *National Institute of Standards and Technology*, US, 2002.
- [4] E. Jones, "An experiential approach to incorporating software testing into the computer science curriculum", *31st ASEE/IEEE FIE '01: Proceedings of the Frontiers in Education Conference*, vol.2, Oct. 2001.
- [5] C. Kaner and S. Padmanabhan, "Practice and transfer of learning in the teaching of software testing," *IEEE Software Engineering Education and Training 2007*, Dublin, Ireland, 2007.
- [6] T. Chen and Pak-Lok Poon, "Experience with teaching Black-box testing in a Computer Science/Software Engineering curriculum," *IEEE Trans. on Education*, vol. 47, no. 1, Feb. 2004, pp. 42-50.
- [7] D.M. Hoffman, P.A. Strooper, and P. Walsh, "Teaching and testing", *9th Conference on Software Engineering Education*, IEEE Computer Society, April 1996.
- [8] A. Bertolino, "Software testing research: Achievements, Challenges, Dreams," *Future of Software Engineering (FOSE '07)*, IEEE press, 2007.
- [9] B. Hetzel, *The Complete Guide to Software Testing*, 2nd Ed., John Wiley and Sons, 1988.
- [10] B. Bezier, *Black-Box Testing*, John Wiley and Sons, 1995.
- [11] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 4th Ed., McGraw Hill, International Edition, 1996.
- [12] L. Copeland, *A practitioner's guide to software testing*, Artech House, London, 2003.
- [13] G.J. Meyers, *The Art of Software Testing*, 2nd Ed. John Wiley and Sons, 2004.
- [14] R. Glass, R. Collard, A. Bertolino, J. Bach, and C. Kaner, "Software testing and industry needs," *IEEE Software*, vol. 23, no. 4, July 2006, pp. 55-57.
- [15] M. Roper, *Software Testing*, McGraw-Hill, London, 1994.
- [16] R. V. Binder, *Testing Object-Oriented system – Models, Patterns and Tools*, Addison Wesley, 1999.
- [17] S. Frezza, "Integrating testing and design methods for undergraduates: teaching software testing in the context of software design," *32nd ASEE/IEEE Frontiers in education conference*, Boston, US, 2002.
- [18] M. Cusumano, A. MacCormack, C. Kemerer, and B. Crandall, "Software development worldwide: the state of the practice," *IEEE Software*, vol. 20, no. 6, Nov. 2003, pp. 28-34.
- [19] J. Miller and C. Mingsins, "Putting the practice in software engineering education," *Proc. of the 1998 International Conference on Software Engineering: Education & Practice*, Los Alamitos, CA, USA, 1998.
- [20] S. Ng, T. Murnane, K. Reed, D. Grant and T. Chen, "A preliminary survey on software testing practices in Australia," *Proc. 2004 Australian Soft. Eng Conf. (ASWEC'04)*, Melbourne, Australia, 2004.
- [21] T. Chan, T. Chen, D. Tang, and T. Tsen, "Software testing education and training in Hong Kong," *Proc. of the 5th International Conference on Quality Software*, Melbourne, Australia, 2005.